LEGO (beyond) toys

Final report Technology Class, 2009

Wouter van Dijk

Table of Contents

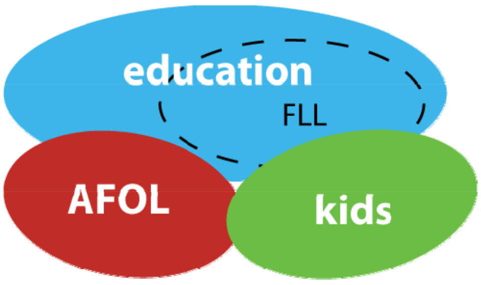
The appendices contain the source code of the demo:

- LEDbrick.java
- Colour.java
- LEDbrickDemo.java

Concept

Target groups

The LEGO Mindstorms products have three main target groups; education, kids and adult fans of LEGO (AFOL's). The first group, using LEGO for educational purposes, is approximately 50% of the Mindstorms market. Its popularity is eminent in the FIRST Lego League, where kids between 8 and 14 years solve a series of missions around a central theme. In 2008, over 130.000 kids worldwide participated in the competition, cooperating with a large number of schools and 45 universities/colleges. Obviously, the educational value of the extension is important to this group. For example, it could indicate measurement values, give status information of the program the robot is running, or keep score in competitions.

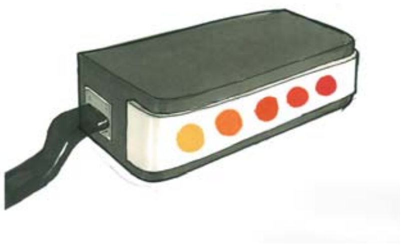


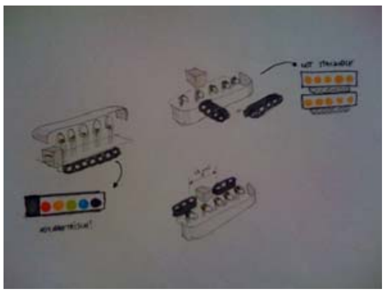
The second group, kids, makes about 25% of the Mindstorms market. For kids to buy (or desire) products, different aspects are important. The extension should have an immediate attractiveness when seen in the store. In use, the quick and easy integration with the other parts is important, as well as support playfulness with the Mindstorms system.

The last group, AFOL's, make the last 25% of the users. This group uses LEGO in combination with other hardware, software, or develops its own extensions. For them, the extension should allow for a wide variety of applications, support expansion and have good specifications.

The LED brick

The proposed extension consists of a brick with five full-colour LED's. The choice for five was partially a practical one; the i^{2}c chip has 16 output channels allowing for 5 x 3-colour LED's. A different option would be to use single colour LED's and use 8 or 16 in the block, but the advantage of being able to use different colours over having more LED's is greater. Having 5 is enough to call it a line or array, have some precision in measurements and allows to use the first and last as eyes of a robot.



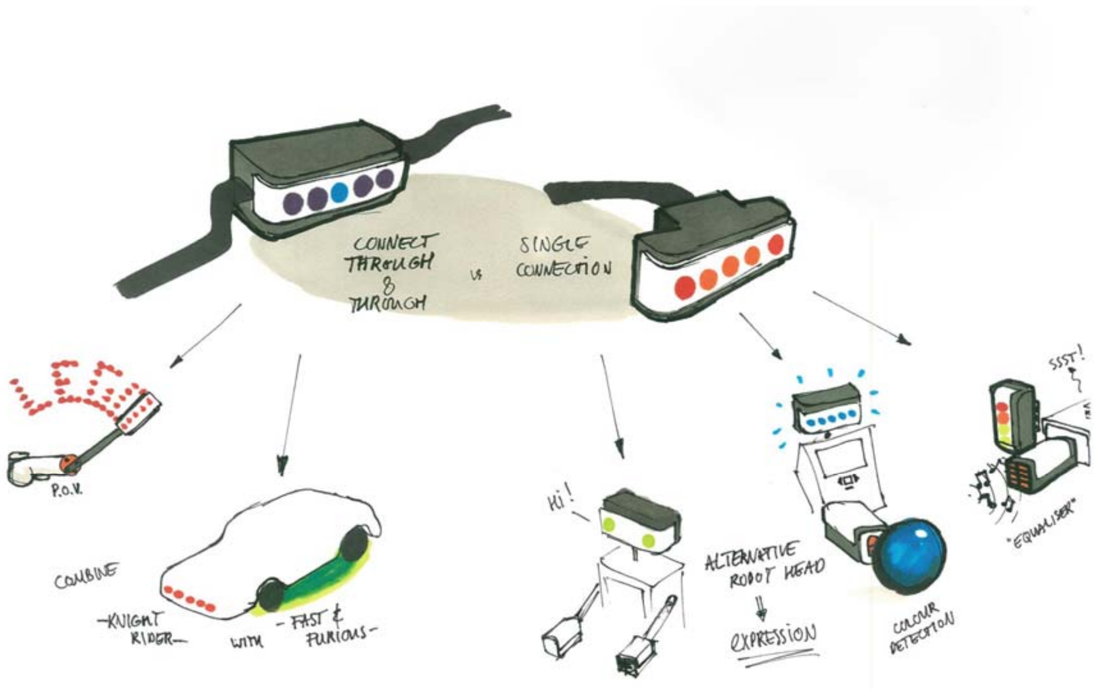


The location of the RJ12 connector and the beams to fix it to a robot were explored briefly. Important aspects were the symmetry of the block when seen from the front and the possibility to stack several blocks. While at first the possibility of connecting several bricks in a daisy chain was explored, this idea was rejected later since a Mindstorms kit would probably have only LED brick due to costs. When users want to add several, they can use different ports or have a special connector. The found solution allows for stacking and is symmetrical. Additionally, it is more compact than other solutions.

Possible applications

With the proposed LED brick extension, several applications can be thought of. A quick list of ideas (some of them also shown in the drawing):

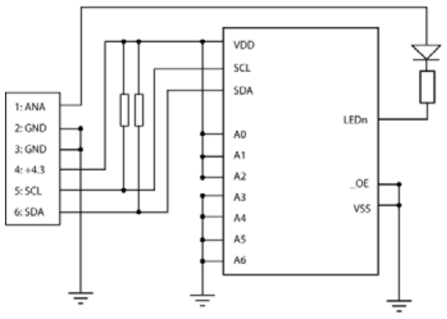
- show the volume measured by the sound sensor
- be a face for robots, possible using colour for expression
- indicate which one of the blue and red balls it recognized
- show the Knight-Rider animation
- flash on collisions
- respond to distance measured by the ultrasonic sensor
- flash like a police siren
- explain binary counting (with touch sensor)
- show messages using persistence of vision (POV)
- general decoration in your favorite colour
- stack several to create a simple display



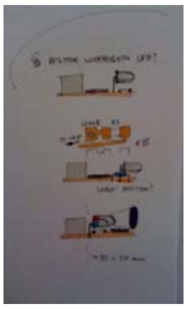
Hardware

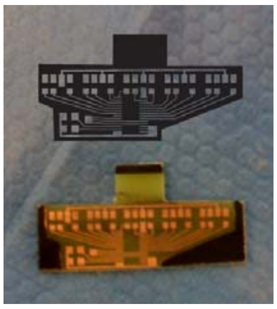
Electronics

The LED brick uses the PCA9635PW chip from NXP as its main element. It is a 16 channel PWM driver with integrated i^{2}c communication functionality. It uses 7 bits for the chip address, which in the prototype is set to 0b0000111 (7). According to LEGO specification, two 82kΩ pull-up resistors are used on the i^{2}c clock and data lines. All LED's are connected to a common supply voltage, not shared with the chip. This allows the usage of a different voltage or power source to drive the LED's. Between the LED and the chip a resistor is used. The red LED's have a 150Ω resistor, the blue and green LED's have a 67Ω resistor. Each LED operates at 20mA, making the maximum power consumption approximately 15 x 20mA = 300mA.



To prevent the brick from becoming too large, the components needed to be placed close to each other. Beside the LED's and resistors, the PCA9635PW chip and especially the RJ12 connector needed a lot of space. After considering several options (see image), a setup was chosen where LED's were placed on the back-side of the PCB, allowing the resistors and chip to be placed underneath the light chambers for the LED's. The connector then would be placed behind the LED's, but the print would not stretch the full width of the print there.

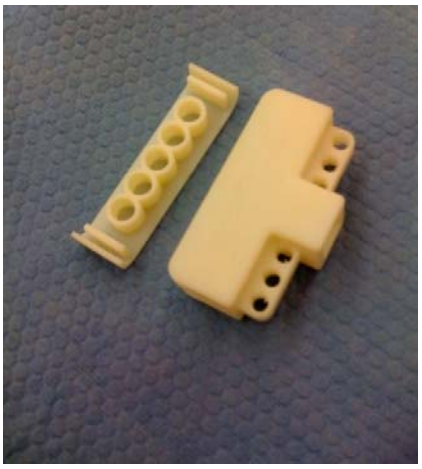




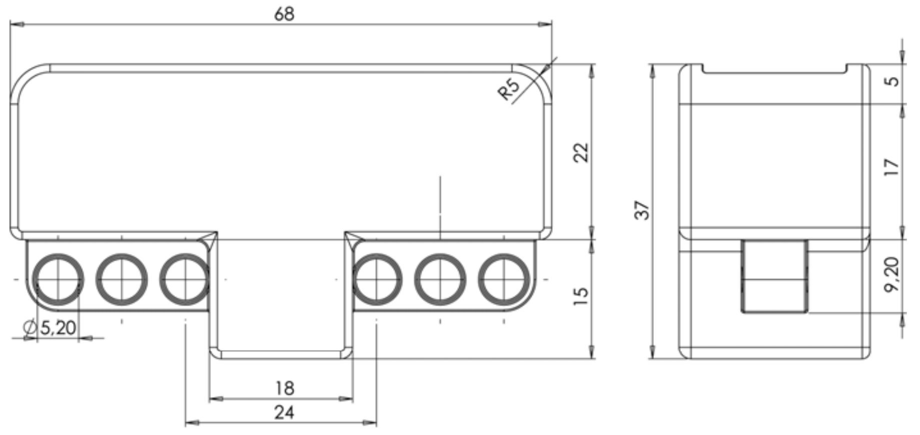
To keep the brick as small as possible and save on soldering wires, a custom PCB was designed using Eagle and Illustrator. After that, the print was made using the laser cutter and etching solution (see image above). *This process will not be described in detail here, if you are interested in knowing more about this process please feel free to contact me.*

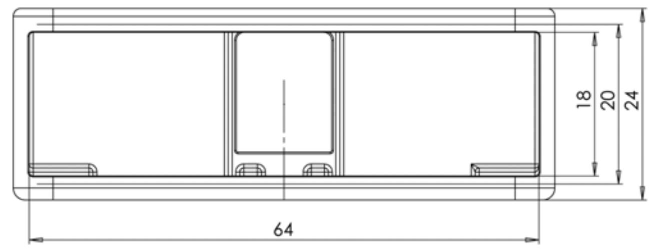
Casing

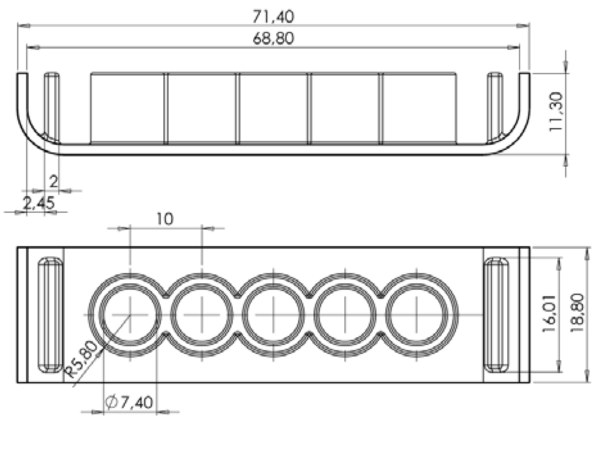
The casing is made up of two parts. The main body contains the print with LED's and the connector. It has a shape like a wide T, with the RJ12 connector in the base of the T. The empty space on both sides of the connector is used for a 3-hole beam to connect the part to a LEGO robot. The second part is a cap that slides on the front of the main body. Like the white cap on other NXT sensors, it has a large rounded corner. On the inside, light chambers are placed to reduce light shining in all directions. In the final model, these light chambers are painted on the inside to reduce internal reflection of the material.

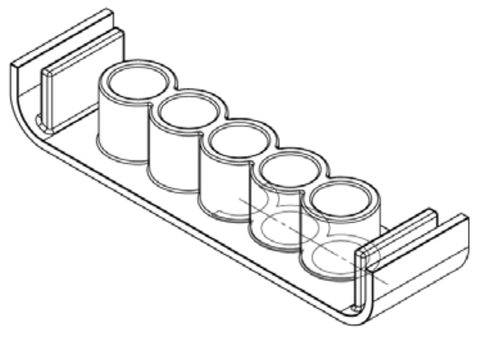


On the right an image of the printed 3D model is shown. On the next page, more detailed drawings with exact measures from the 3D model.


68
R5
22
15
∅5,20
18
24
37
5
17
9,20

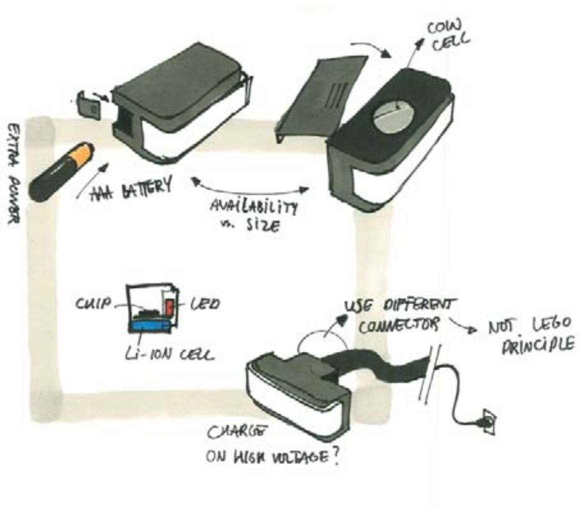

18
20
24
64


71,40
68,80
11,30
2
2,45
10
16,01
18,80
R5,80
∅7,40



Power consumption

In the current NXT brick, each sensor port can use a maximum of 20 mA. This is by far not enough for five RGB LED's, which are in effect 15 LED's. The motor outputs A, B and C are capable of providing 700 mA at 9V, but i^{2}c is not possible on these ports.



Three possible alternatives to provide power are replaceable AA or AAA batteries, a replaceable coin cell or a (Li-ion) power cell that can be recharged. However, they all make the LED brick significantly larger and make both the usage and casing a lot more complex.

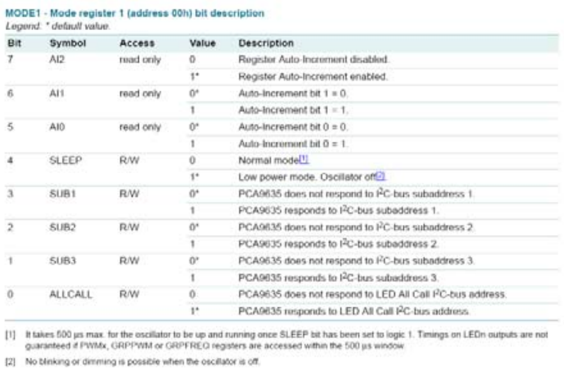
Since the extension is a concept and does not necessarily have to work on the current NXT brick, a change in the NXT hardware is anticipated where high power and i^{2}c communication are made possible through a single port, which would probably be a motor/actuator port.

Software

PCA9635PW configuration

The NXP chip used, the PCA9635PW, is a 16 channel LED driver using pulse width modulation at a frequency of 97 Hz. Each channel has a resolution of 1 bit; 256 steps. The i^{2}c communication protocol is integrated in the chip, making it ideal for this application.

On initialization of the chip, some settings need to be adjusted to make optimal use of the chip. Two tables from the datasheet showing the possible settings:



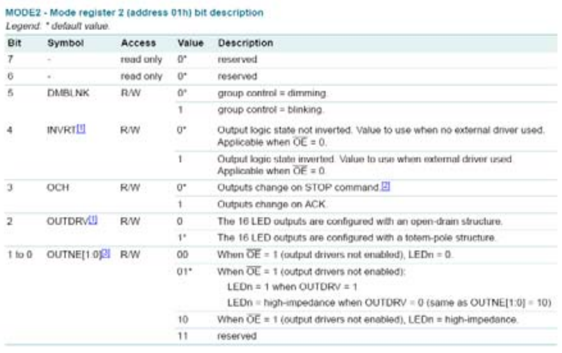
MODE1 - Mode register 1 (address 00h) bit description

*Legend: * default value.*

Bit	Symbol	Access	Value	Description
7	AI2	read only	0	Register Auto-Increment disabled.
			1*	Register Auto-Increment enabled.
6	AI1	read only	0*	Auto-Increment bit 1 = 0.
			1	Auto-Increment bit 1 = 1.
5	AI0	read only	0*	Auto-Increment bit 0 = 0.
			1	Auto-Increment bit 0 = 1.
4	SLEEP	R/W	0	Normal mode[1].
			1*	Low power mode. Oscillator off[2].
3	SUB1	R/W	0*	PCA9635 does not respond to I^{2}C-bus subaddress 1.
			1	PCA9635 responds to I^{2}C-bus subaddress 1.
2	SUB2	R/W	0*	PCA9635 does not respond to I^{2}C-bus subaddress 2.
			1	PCA9635 responds to I^{2}C-bus subaddress 2.
1	SUB3	R/W	0*	PCA9635 does not respond to I^{2}C-bus subaddress 3.
			1	PCA9635 responds to I^{2}C-bus subaddress 3.
0	ALLCALL	R/W	0	PCA9635 does not respond to LED All Call I^{2}C-bus address.
			1*	PCA9635 responds to LED All Call I^{2}C-bus address.

[1] It takes 500 μs max. for the oscillator to be up and running once SLEEP bit has been set to logic 1. Timings on LEDn outputs are not guaranteed if PWMx, GRPPWM or GRPFREQ registers are accessed within the 500 μs window.

[2] No blinking or dimming is possible when the oscillator is off.



MODE2 - Mode register 2 (address 01h) bit description

*Legend: * default value.*

Bit	Symbol	Access	Value	Description
7	-	read only	0*	reserved
6	-	read only	0*	reserved
5	DMBLNK	R/W	0*	group control = dimming.
			1	group control = blinking.
4	INVRT[1]	R/W	0*	Output logic state not inverted. Value to use when no external driver used. Applicable when $\overline{OE}$ = 0.
			1	Output logic state inverted. Value to use when external driver used. Applicable when $\overline{OE}$ = 0.
3	OCH	R/W	0*	Outputs change on STOP command.[2]
			1	Outputs change on ACK.
2	OUTDRV[1]	R/W	0	The 16 LED outputs are configured with an open-drain structure.
			1*	The 16 LED outputs are configured with a totem-pole structure.
1 to 0	OUTNE[1:0][3]	R/W	00	When $\overline{OE}$ = 1 (output drivers not enabled), LEDn = 0.
			01*	When $\overline{OE}$ = 1 (output drivers not enabled): LEDn = 1 when OUTDRV = 1; LEDn = high-impedance when OUTDRV = 0 (same as OUTNE[1:0] = 10)
			10	When $\overline{OE}$ = 1 (output drivers not enabled), LEDn = high-impedance.
			11	reserved

LeJOS implementation

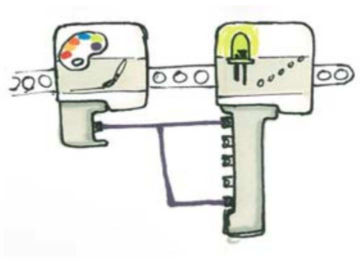
Since the chip in the LED brick uses the i^{2}c protocol, all communication could be done with the LeJOS function *I2CSensor.sendData*, requiring only an address and a value.

In initialization of the program, the mode registers are set to the right state. Then, the LED's can be set by calling their exact address. The chip also allows auto-incrementing the register addresses, making it possible to call the first LED's address and then sending the values for all LED's directly. Unfortunately, the buffer of the NXT brick is too small to contain values for all LED's so this function could not be used.

In order to make writing the code for the brick easier, a very simple Java color class was written. The class contains values for red, green and blue and has simple set and get functions. On initialization of the LED brick, some basic colors are created (red, orange, green, cyan, blue, purple).

The main class that controls the LED brick contains some functions that allow setting LED's, contains the basic colors and implements the communication to the chip in the brick.

NXT-G



In the graphical programming environment of LEGO Mindstorms called NXT-G, two parts would need to be added for controlling the LED brick. To enable the use of all colors, a 'Color' block need to be designed. This would have just one output containing a color, which can be set using a palette available in the block's properties.

The most important part would be the LED block itself. Depending on the mode, it could contain five inputs that require a color variable. This is then directly translated into light on the brick. This would work for all color functions that are not animated. For animated applications, the block could contain for example a color, a frequency and a direction.

```java
import lejos.nxt.*;

public class LEDbrick {

    // R > L, B-R-G

    public static final Colour ON = new Colour(255,150,150);
    public static final Colour OFF = new Colour(0,0,0);
    public static final Colour GREEN = new Colour(0,150,0);
    public static final Colour ORANGE = new Colour(255,10,0);
    public static final Colour RED = new Colour(255,0,0);
    public static final Colour BLUE = new Colour(0,0,255);
    public static final Colour CYAN = new Colour(0,255,255);
    public static final Colour PURPLE = new Colour(255,0,255);

    int address = 7;
    I2CSensor brick;
    float[] LEDval;
    byte[] goal;

    public LEDbrick(I2CPort thePort) {

        brick = new I2CSensor(thePort);
        brick.setAddress(address);

        initLEDbrick();
        LEDval = new float[16];
        goal = new byte[16];

    }

    public void setLED(int LEDnum, Colour c) // LED# 1-5
    {
        int i = ((LEDnum-1)*3);
        goal[i] = c.b;
        goal[i+1] = c.r;
        goal[i+2] = c.g;
    }
    public void setAll(Colour c)
    {
        for (int i = 1; i <= 5; i++)
        {
            setLED(i,c);
        }
    }
    public void setTo(int to,Colour c)
    {
        for (int i = 1; i <= 5; i++)
        {
            setLED(i,((i <= to) ? c : OFF));
        }
    }
    public void setFace(Colour c)
    {
        setLED(1,c);
        setLED(2,OFF);
        setLED(3,OFF);
        setLED(4,OFF);
        setLED(5,c);
    }
    public void clearAll()
    {
        for (int i = 0; i < goal.length; i++)
        {
            goal[i] = (byte)0;
        }
    }
    private void initLEDbrick()
    {
        // set modes
        brick.sendData(0x00, (byte) 0x80);
        brick.sendData(0x01, (byte) 0x0e);
        brick.sendData(0x02, (byte) 0x00);
        // set LEDOUT states
        brick.sendData(0x14, (byte) 0xAA); // 0b10101010
        brick.sendData(0x15, (byte) 0xAA);
        brick.sendData(0x16, (byte) 0xAA);
        brick.sendData(0x17, (byte) 0xAA);
    }

    public void updateLEDs()
    {
        // calc new LED values & write to brick
        for (int i = 0; i < 16; i++)
        {
            //LEDval[i] += ((float)goal[i]-LEDval[i])/(float)2.0;
            //if (goal[i]-LEDval[i] != 0)
            //{ LEDval[i] += (goal[i]-LEDval[i] > 0) ? 0.1 : -0.1; }
            brick.sendData(2+i,(byte)goal[i]);
        }
    }

}
```

```java
public class Colour {
    public byte r;
    public byte g;
    public byte b;
    Colour(int a, int b, int c)
    {
        this.r = (byte)a;
        this.g = (byte)b;
        this.b = (byte)c;
    }
    public void set(Colour c)
    {
        this.r = (byte)c.r;
        this.g = (byte)c.g;
        this.b = (byte)c.b;
    }
}
```

```java
import lejos.nxt.*;

public class LEDbrickDemo implements TimerListener  {

    // initialize the LEDbrick with address 7 at port S4
    public static LEDbrick l = new LEDbrick((I2CPort)SensorPort.S4);
    public static int num = 3;
    public static Timer t;
    public static long nextPress = 0;
    public static long nextKR = 0;
    public static int bin = 0;
    public static boolean binCounting = true;
    public static int mode = 0;
    public static TouchSensor touch;
    public static SoundSensor sound;
    public static UltrasonicSensor ultra;
    public static LightSensor light;
    public static int kr = 1;
    public static int krdir = 1;
    public static int count = 0;
    public static int police = 0;
    public static int face = 0;

    public static void main(String[] args) throws InterruptedException
    {
        Motor.A.stop();

        l.clearAll();
        touch = new TouchSensor(SensorPort.S1);
        sound = new SoundSensor(SensorPort.S2);
        //ultra = new UltrasonicSensor(SensorPort.S3);
        light = new LightSensor(SensorPort.S3);
        light.setFloodlight(false);

        t = new Timer(50,new LEDbrickDemo());
        //t.start();

        LCD.drawString("ESCAPE stops", 0,7);
        mode = 7;

        while (!Button.ESCAPE.isPressed())
        {

            LCD.drawString(""+mode, 0,0);
            if (Button.RIGHT.isPressed() && System.currentTimeMillis() > nextPress)
            {
                //t.start();
                count = 0;
                mode++;
                if (mode == 8) { mode = 1; }
                //if (mode == 1) { Motor.A.rotateTo(-90, true); }
                //if (mode == 4) { Motor.A.rotateTo(0, true); }
                //if (mode == 5) { t.stop(); }
                nextPress = System.currentTimeMillis()+200;
            }


            if (mode == 1) // sound sensor
            {
                int vol = sound.readValue();
                l.setLED(5,l.GREEN);
                l.setLED(4,((vol > 20) ? l.GREEN : l.OFF));
                l.setLED(3,((vol > 40) ? l.ORANGE : l.OFF));
                l.setLED(2,((vol > 60) ? l.ORANGE : l.OFF));
                l.setLED(1,((vol > 80) ? l.RED : l.OFF));
                l.updateLEDs();
                LCD.drawString("vol:"+vol, 0,3);
            }
            else if (mode == 2) // light sensor
            {
                int lum = light.readNormalizedValue();
                if (lum < 200) { l.setAll(l.BLUE); }
                else if (lum > 250) { l.setAll(l.OFF); }
                else { l.setAll(l.RED); }
                l.updateLEDs();
                LCD.drawString("lum:"+lum, 0,3);
            }/*
            else if (mode == 2) // distance sensor
            {
                int ult = ultra.getDistance();
                LCD.drawString("ult:"+ult+"   ", 0,3);
                l.setLED(5,l.CYAN);
                l.setLED(4,((ult > 100) ? l.CYAN : l.OFF));
                l.setLED(3,((ult > 200) ? l.CYAN : l.OFF));
                l.setLED(2,((ult > 300) ? l.CYAN : l.OFF));
                l.setLED(1,((ult > 400) ? l.CYAN : l.OFF));
                l.updateLEDs();
            }*/
            else if (mode == 3 && System.currentTimeMillis() > nextPress) // touch ding
            {
                l.clearAll();
                if (touch.isPressed()) { count++; }
                l.setLED(1, ((count == 1 || count == 3 || count == 5 || count == 7) ? 
l.CYAN : l.OFF));
                l.setLED(2, ((count == 2 || count == 3 || count == 6 || count == 7) ? 
l.CYAN : l.OFF));
                l.setLED(3, ((count > 3 && count < 8) ? l.CYAN : l.OFF));
                l.setLED(4, ((count == 8) ? l.CYAN : l.OFF));

                l.updateLEDs();
                nextPress = System.currentTimeMillis()+100;
            }
            else if (mode == 4) // knight rider
            {
                if (touch.isPressed())
                {
                    l.setAll(l.RED);
                }
                else if (System.currentTimeMillis() > nextKR)
                {
                    kr+=krdir;
                    LCD.drawString("kr:"+kr+"   ", 0,3);
                    for (int i = 1; i < 6; i++)
                    {
                        l.setLED(i,((i == kr) ? l.RED : l.OFF));
                    }
                    if (kr == 1 || kr == 5) { krdir *= -1; }
                    nextKR = System.currentTimeMillis()+60;
                }
                l.updateLEDs();
            }
            else if (mode == 5 && System.currentTimeMillis() > nextPress) // random
            {
                Colour nu;
                double r = Math.random();
                if (r < 0.16) { nu=l.RED; }
                else if (r < 0.33) { nu=l.PURPLE; }
                else if (r < 0.5) { nu=l.CYAN; }
                else if (r < 0.66) { nu=l.ORANGE; }
                else if (r < 0.83) { nu=l.BLUE; }
                else { nu=l.GREEN; }
                num = (Math.random() < 0.5) ? Math.max(num-1, 1) : Math.min(num+1, 5);
                l.clearAll();
                for (int i = 1; i < 6; i++)
                {
                    l.setLED(num, nu);
                }
                l.updateLEDs();
                nextPress = System.currentTimeMillis()+((int)(Math.random()*100))+50;
            }
            else if (mode == 6 && System.currentTimeMillis() > nextPress)
            {
                l.clearAll();
                if (police == 0)
                {
                    l.setLED(1, l.RED);
                    l.setLED(2, l.RED);
                }
                else if (police == 1)
                {
                    l.setLED(4, l.BLUE);
                    l.setLED(5, l.BLUE);
                }
                l.updateLEDs();
                police++;
                if (police == 2) { police = 0; }
                nextPress = System.currentTimeMillis()+130;
            }
            else if (mode == 7)
            {
                if (face == 0) { l.setFace(l.GREEN); }
                else if (face == 1) { l.setFace(l.CYAN); }
                else if (face == 2) { l.setFace(l.RED); }
                else if (face == 3) { l.setFace(l.ORANGE); }
                l.updateLEDs();
                if (touch.isPressed() && System.currentTimeMillis() > nextPress)
                {
                    face++;
                    if (face == 4) { face = 0; }
                    nextPress = System.currentTimeMillis()+200;
                }
            }
        }

    }

    public void timedOut()
    {
        l.updateLEDs();
    }
}
```